SYSTEMS AND METHODS FOR READING ONLY DURABLY COMMITTED DATA IN A SYSTEM THAT OTHERWISE PERMITS LAZY COMMIT OF TRANSACTIONS

Technical Field

[0001] The present invention relates generally to database transactions, and more particularly, to enabling efficient "durable read" capabilities to enable proper isolation of transactions from the effects of lazy commit transactions in a data base system that utilizes a transaction log to ensure data consistency in the event of an unexpected system interruption and allows "lazy commits."

Background

[0002] A "lazy commit" allows database transactions to be committed faster by not waiting for log records of the transaction to be synchronously written to disk. When a system crash occurs, transactions may be lost because the log records are not available to redo the transaction. For certain applications, that is acceptable because these applications can recreate the transactions after resumption from a crash. While lazy commits speed up processing for this class of applications, they also create the potential for data inconsistency in case another application reads "lazily committed" changes from a first store and updates a second store based on that data. In the event of a crash, the changes in the first store might be lost, leaving the first store and the second store in an inconsistent state. What is missing in the art is an efficient

means for certain transactions to conduct "durable reads"—that is, read only "durably committed" data, to isolate their processing from that of a lazy commit transaction to eliminate the possibility of the above-mentioned inconsistency. The present invention provides a solution.

Summary

[0003] Various embodiments of the present invention enable "durable reads" for transactions that require isolation from the effects of lazy commit transactions and which must be guaranteed to read only durably committed data. When a durable read transaction attempts to read data changed by a lazy commit transaction, the system ensures the lazy commit transaction's changes are first made durable.

[0004] In one embodiment of the present invention, a data page is marked (as "not durable") after a "lazy commit" transaction makes changes to the data page. Then, when a second transaction seeking to obtain durable data from the changed data page determines that the data page is marked (that the data is not durable), the transaction causes the log to immediately flush to the disk so that the commit log entry that pertains to the "lazy commit" transaction that modified the data page becomes a durable log entry. The transaction also causes the "lazy commit" transactions that have not yet committed but which might have modified the page to flush their transaction log entries to disk whenever they commit. The data page is then unmarked (immediately or at a later point in time) and the data, now durable (because of flushing the log entry to the persistent data store), is read from the data page by the durable read transaction.

Brief Description Of The Drawings

[0005] The foregoing summary, as well as the following detailed description, is better understood when read in conjunction with the appended drawings. For the purpose of illustrating the invention, there is shown in the drawings exemplary constructions of the invention; however, the invention is not limited to the specific methods and instrumentalities disclosed. In the drawings:

[0006] Fig. 1 is a block diagram representing a computer system in which aspects of the present invention may be incorporated;

[0007] Fig. 2 is a block diagram illustrating the metaphorical framework for one embodiment of a strong ACID-based transaction manager system employing a "durable commit" strategy;

[0008] Fig. 3 is a block diagram that illustrates an alternative to the careful write approach using a "lazy commit" strategy.

[0009] Figs. 4 is a block diagrams that illustrates the method of various embodiments of the present invention to provide a durable read capability in a system that permits lazy commit transactions.

[0010] Fig. 5 is a flowchart illustrating the method by which data pages are marked as "not durable" by a lazy commit transaction.

[0011] Fig. 6 is a flowchart illustrating the method by which the log is flushed to disk in order to provide an application with the ability to make a durable read of a data page changed by a lazy commit transaction.

[0012] Fig. 7 is a flowchart illustrating the method by which the data pages are unmarked in one embodiment of this invention.

Detailed Description

[0013] The subject matter is described with specificity to meet statutory requirements. However, the description itself is not intended to limit the scope of this patent. Rather, the inventors have contemplated that the claimed subject matter might also be embodied in other ways, to include different steps or elements, or combinations thereof, similar to the ones described in this document, in conjunction with other present or future technologies. Moreover, although the term "step" may be used herein to connote different elements of methods employed, the term should not be interpreted as implying any particular order among or between various steps herein disclosed unless and except when the order of individual steps is explicitly described.

Computer Environment

[0014] Numerous embodiments of the present invention may execute on a computer. Fig. 1 and the following discussion is intended to provide a brief general description of a suitable computing environment in which the invention may be implemented. Although not required, the invention will be described in the general context of computer executable instructions, such as program modules, being executed by a computer, such as a client workstation or a server. Generally, program modules include routines, programs, objects, components, data structures and the like that perform particular tasks or implement particular abstract data types. Moreover, the invention may be practiced with other computer system configurations, including hand held devices, multi processor systems, microprocessor based or programmable consumer electronics, network PCs, minicomputers, mainframe computers and the like. The invention may also be practiced in distributed computing environments where tasks are performed by remote

processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

[0015] As shown in Fig. 1, an exemplary general purpose computing system includes a conventional personal computer 20 or the like, including a processing unit 21, a system memory 22, and a system bus 23 that couples various system components including the system memory to the processing unit 21. The system bus 23 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. The system memory includes read only memory (ROM) 24 and random access memory (RAM) 25. A basic input/output system 26 (BIOS), containing the basic routines that help to transfer information between elements within the personal computer 20, such as during start up, is stored in ROM 24. The personal computer 20 may further include a hard disk drive 27 for reading from and writing to a hard disk, not shown, a magnetic disk drive 28 for reading from or writing to a removable magnetic disk 29, and an optical disk drive 30 for reading from or writing to a removable optical disk 31 such as a CD ROM or other optical media. The hard disk drive 27, magnetic disk drive 28, and optical disk drive 30 are connected to the system bus 23 by a hard disk drive interface 32, a magnetic disk drive interface 33, and an optical drive interface 34, respectively. The drives and their associated computer readable media provide non volatile storage of computer readable instructions, data structures, program modules and other data for the personal computer 20. Although the exemplary environment described herein employs a hard disk, a removable magnetic disk 29 and a removable optical disk 31, other types of computer readable media which can store data that is accessible by a computer, such as magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, random access

memories (RAMs), read only memories (ROMs) and the like may also be used in the exemplary operating environment.

[0016] A number of program modules may be stored on the hard disk, magnetic disk 29, optical disk 31, ROM 24 or RAM 25, including an operating system 35, one or more application programs 36, other program modules 37 and program data 38. A user may enter commands and information into the personal computer 20 through input devices such as a keyboard 40 and pointing device 42. Other input devices (not shown) may include a microphone, joystick, game pad, satellite disk, scanner or the like. These and other input devices are often connected to the processing unit 21 through a serial port interface 46 that is coupled to the system bus, but may be connected by other interfaces, such as a parallel port, game port or universal serial bus (USB). A monitor 47 or other type of display device is also connected to the system bus 23 via an interface, such as a video adapter 48. In addition to the monitor 47, personal computers typically include other peripheral output devices (not shown), such as speakers and printers. The exemplary system of Fig. 1 also includes a host adapter 55, Small Computer System Interface (SCSI) bus 56, and an external storage device 62 connected to the SCSI bus 56.

[0017] The personal computer 20 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 49. The remote computer 49 may be another personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the personal computer 20, although only a memory storage device 50 has been illustrated in Fig. 1. The logical connections depicted in Fig. 1 include a local area network

(LAN) 51 and a wide area network (WAN) 52. Such networking environments are commonplace in offices, enterprise wide computer networks, intranets and the Internet.

[0018] When used in a LAN networking environment, the personal computer 20 is connected to the LAN 51 through a network interface or adapter 53. When used in a WAN networking environment, the personal computer 20 typically includes a modem 54 or other means for establishing communications over the wide area network 52, such as the Internet. The modem 54, which may be internal or external, is connected to the system bus 23 via the serial port interface 46. In a networked environment, program modules depicted relative to the personal computer 20, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

[0019] While it is envisioned that numerous embodiments of the present invention are particularly well-suited for computerized systems, nothing in this document is intended to limit the invention to such embodiments. On the contrary, as used herein the term "computer system" is intended to encompass any and all devices capable of storing and processing information and/or capable of using the stored information to control the behavior or execution of the device itself, regardless of whether such devices are electronic, mechanical, logical, or virtual in nature.

Transactions

[0020] In regard to databases and other information storage structures, a transaction is a sequence of information exchanges and related work that are treated as a single unit for purposes of satisfying a request and for ensuring database integrity. In order to ensure integrity, a transaction is deemed complete or "committed" only when the transaction commit is recorded in

a transaction log that is written to a persistent data store. This commit log record is written to the persistent data store prior to the changed data resulting from the transaction being written to the persistent data store. Should something happen after the transaction is committed but before the data resulting from the transaction is successfully recorded to the persistent data store—that is, before the data itself is stored in the persistent data store, but after the commit log has been recorded to the data store—changes recorded in the log can be used to bring the data in the data store up to date to a state corresponding to that reflected in the transaction log.

[0021] A transaction manager is a program/component that manages or oversees the sequence of events that comprise a transaction. Transactions are supported by Structured Query Language (SQL), the standard user and programming interface for databases. The four primary attributes for any transaction made by a transaction manager are atomicity, consistency, isolation, and durability (ACID). For a transaction involving two or more discrete pieces of information, atomicity is the requirement that all pieces of information must be committed in order for the transaction itself to be deemed committed; otherwise, a transaction is deemed uncommitted. To ensure consistency, a transaction either creates a new and valid state of data, or, if any failure occurs, returns all data to its state before the transaction was started. A transaction in process and not yet committed must remain isolated from any other transactions. Finally, committed data must be saved by the system such that, even in the event of a failure and system restart, the data is available in a correct state. The ACID concept is described in ISO/IEC 10026-1:1992 Section 4, and each of the ACID attributes is generally measured against a benchmark.

[0022] Fig. 2 is a block diagram illustrating the metaphorical framework for one embodiment of a strong ACID-based database system employing a "durable commit" strategy.

An application 202, which may be just one from among a plurality of applications 212, makes changes 222 to a data page 204 (step 1), which may be one from among a plurality of data pages 214 in volatile memory 200. These changes to the data page 204 are part of a transaction (not shown) that are not immediately written 224 to the persistent data store 208. Instead, the updated data pages 204 are written 224 at a later point, on certain occasions, to the persistent data store 208. Of course, if there is a system crash, these changes would be lost since they are not reflected in the data store 208; therefore, the data manager also maintains a transaction log (not shown) in the data store 208.

[0023] For every change that is written to any of the data pages 214, a corresponding log record describing the change is written 230 to the transaction log buffer 210 (step 2). Every log record generated is given a sequence number referred to as a Log Sequence Number (LSN). This LSN is also written 248 to the data page 204 in an attribute called Page LSN 250 (step 3). Page LSN means the LSN of the last log record corresponding to the last change made to the page.

[0024] When the transaction is requested to be committed by the application 202, a commit log record (not shown) is written to the transaction log buffer 210. Then the transaction log buffer 210 is then written 232 to the persistent data store 208 (step 4) before the commit request is considered completed. It is the writing of the commit log record to the persistent data store 208 that ensures the durability of the changes made as part of this transaction. Then, at a later point, the data page 204 is itself written 224 to the persistent data store 208 (step 5).

[0025] At any point, should the computer system crash and subsequently reboot, some changes to the data pages may not have not written to the persistent data store (step 5) at the time of the crash although the corresponding commit log entry was so written (step 4). The data

manager, referencing the transaction log in the persistent data store, can determine the present state of the durable data by ascertaining which transactions were committed and which were not. After determining which transactions were committed and which data pages do not have the changes corresponding to those transactions, the data manager re-applies the changes described in the transaction log to those data pages and then writes them to the persistent data store. This makes sure that none of the changes to data pages performed by an application in context of a committed transaction are lost in case of a crash. Therefore, a transaction is considered to be committed if a log entry has been flushed to the data store regardless of whether the actual data page is actually stored in the data store before a crash or other such events.

[0026] While this approach provides a means for ensuring strong data consistency, the required serial and synchronous writes of transaction log to the persistent data store are extremely time-consuming and resource costly. The durability of a transaction is achieved by flushing the transaction log buffer 210 to the transaction log in the persistent data store 208 at the time of transaction commit. However, this flushing is quite expensive because most persistent data stores have high latency and low throughput.

Lazy Commit

[0027] An alternative to the durable commit approach is the "lazy commit" approach. Applications can achieve significant performance, latency, and throughput improvements by indicating to the data manager that the immediate durability of the committed transactions in the event of a crash is not a requirement for them. This would allow these applications to commit a larger number of transactions in a given amount of time than would have been otherwise possible.

[0028] There are many applications where the durability (out of Atomicity, Consistency, Isolation, Durability) of a transaction in the event of a crash is not required by an application but the performance is very important. The examples of such applications include

1. Data Warehousing
2. Queue Processing
3. Bulk Load

[0029] These applications are designed such that, after resuming from a crash, they can reconstruct the transactions that were lost due to the use of lazy commit (followed by system crash). For example, bulk load can store the current location in the input file up to which the data has been loaded into the database, as part of the transaction that is inserting the data. If some of the transactions were lost because of a system crash, the file position stored in the database would indicate the location from which the data should be loaded from the input file. Hence, for bulk load, "lazy commit" provides significant performance gains and there is no loss of data in case of system crash.

[0030] However, while this lazy-commit approach provides a more efficient means for conducting transactions and ensuring fairly strong data consistency (it provides Atomicity, Consistency, Isolation, and delayed Durability), one serious shortcoming of this method is that it allows other applications to read and operate on data that has been modified by a committed transaction but which might not yet be durable, that is, where the corresponding commit log record in the transaction log buffer has not yet been flushed to the persistent data store (step 4); consequently, if the system crashes before the corresponding entry in the transaction log is flushed to the data store, an inconsistency can result in certain specific situations.

[0031] For example, and in reference to Fig. 3, one such inconsistency that can arise is as follows: An application 202 indicates to the data manager to perform lazy commit of a transaction, and thus the commit 232 (step 4) does not occur before other applications are allowed to access the data in the changed data page 204. Another application 202' then reads 262 the data on the modified data page 202 (step 3a) and writes 224' that data to a second persistent data store 208' (step 3b). If a system crash occurs at this point (before steps 4 and 5 as shown), the commit log record of the transaction will not have been written 232 to the transaction log in the persistent data store, and thus all the changes made as part of transaction would be lost. However, it is possible that the transaction for the second application 202' in the second data store 208' may have durably committed, and this would lead to an inconsistency where the changes are present in the second data store 208' but are missing from the first data store 208.

[0032] What is missing in the art is an effective and efficient means by which an application in a system that allows lazy-commits to make a "durable read" of data (that is, read data that is both committed and logged to the data store 208). While certain inefficient means do exist (such as flushing all commit logs and/or suspending all lazy commit transactions whenever a durable read application is processing), these techniques are inadequate and the various embodiments of the present invention provide an alternative that enables a system to gain most of the benefits of using a lazy commit approach while also providing a durable read capability.

Durable Read

[0033] In one embodiment of the present invention, and as illustrated in Fig. 4, a "lazy commit" application 202 makes changes 222 to a data page 204 (step 1) and marks 206 the data

page 204 as potentially "not durable" (step 2). This can be achieved by, for example, marking a single bit reserved for this purpose on the page, among other diverse methodologies. The changed data page 204 is not immediately written to the persistent data store 208 but a log entry 230 is made to the transaction log buffer 210 describing this change (step 3) and the Page LSN 210 is also written 248 to the data page 204 (step 4) as described in the background section above. When the application requests the transaction to be committed, a commit log record is written to the transaction log buffer 210 but the transaction log buffers 210 are not flushed to the transaction log in the persistent data store 208 as described in the Lazy commit section above. The application 202 continues its processing under the assumption that the transaction is committed and will become durable at some point in time in future (commit log record written to the persistent data store 208).

[0034] Given this present state, a second application 202' seeking to obtain durable data from the changed data page 204 first determines 268 if the data page 204 is marked 206 (step 5) which, in this case, it is (and therefore potentially has non-durable data). The application 202' (directly, via the lazy commit application 202, via the transaction manager, or otherwise) causes the transaction log buffers 210 to immediately flush 232 to the data store 208 so that the commit log entry that pertains to the lazy commit transaction becomes a durable log entry in the data store 208 (step 6) and unmarks the data page (not shown). The application 204' can then read 262 the data from the data page 204 (step 7) and store 224' said data in its data store 208' (step 8). Of course, flushing (step 6) is not required if the data on this page has already been guaranteed durable by another durable read application. The method for doing that is demonstrated in figure 6 and is described later in this document.

[0035] In general, if the lazy commit transaction that modified this data page has not yet committed, actions are taken to ensure that whenever that transaction commits it flushes the commit log record to the transaction log in the persistent data store (that is, it performs the durable commit and not the lazy commit). One of the ways this can be done is by increasing a store-wide variable called "DurableCommitLSN" to the PageLSN of this page. Whenever a lazy commit transaction is commited, the BeginLSN of that transaction is compared to the DurableCommitLSN and if the BeginLSN of this transaction is less than the DurableCommitLSN, then, this transaction might have modified the data page in question, and hence, flushes its commit log record to the transaction log in the durable data store D. If a second application is reading the same row that a first application has modified, the second application has to wait, after setting DurableCommitLSN , for the first application to commit and release locks before it can read the row.

[0036] The modified data page is thus guaranteed to have changes that are durable and the data, now durable, is read from the data page by the durable read application. The data manager or the applications store the information that this data page contains durable data so that any application that intends to do durable read for this page at any time in future, does not have to flush the transaction log buffers. Two of the ways in which this can be achieved is as follows:

1. The bit on the page that is used to mark this page as potentially containing non-durable data can be cleared. On some systems, doing it using the method above might have some undesirable performance consequences. For example, a reader typically acquires read latch on the page. If the reader were to clear this bit, it would have to acquire an exclusive latch and mark the page dirty. That would reduce concurrency and increase the number of I/O's in the system.

2. A value "DurableReadLSN" is maintained by the system. This value indicates that all data pages with pageLSN < DurableReadLSN have only durable data. Whenever a durable read application takes actions to make data on a data page durable, it can increment the DurableReadLSN to the pageLSN of the page that was made durable. Subsequent durable reads by applications compare the DurableReadLSN and pageLSN and take actions to make the data page durable only if the pageLSN > DurableReadLSN. This method has the problem of the marking bit on the page never getting cleared. This problem is resolved by comparing the pageLSN of every page that is written to the persistent data store with the DurableReadLSN. If the pageLSN < DurableReadLSN, then the bit can be cleared just before writing the data page to the persistent data store.

[0037] In various embodiments of the present invention, any or all of the actions described herein may be conducted by the application itself, by another application, by the system manager, or by another means, and nothing herein is intended to limit execution of each step in the methodologies to any particular component. With this in mind, one embodiment of the present invention where actions are performed by the transaction manager is herein described.

[0038] Referring to Fig. 5, a transaction (lazy commit or durable commit) is performed upon entry 500 wherein, at initial step 501, changes are made to the subject data page in memory. At step 504, the transaction manager determines whether the transaction is a lazy commit transaction and, if so, at step 506, the transaction manager marks the data page (to indicate that it is not durable). The steps 501 to 506 might be repeated multiple times with different subject data pages within the same transaction. At step 508, the application requests the

transaction to be committed. The transaction manager determines whether this is a lazy commit transaction 510. If it is, the commit log record is written to the transaction log buffer but the buffer is not flushed to the persistent data store. If the transaction manager finds that a durable reader has requested this transaction to be durably committed 512 or , if the transaction is not a lazy commit transaction (in which case it is a "durable commit" transaction), the transaction manager immediately flushes the commit log at step 514 and then returns at step 516.

[0039] Referring to Fig. 6, a read transaction (durable read or non-durable read) is performed upon entry 600 wherein, at initial step 602, the transaction manager determines if the read is to be a durable read and, if so, at step 606, the transaction manager further determines if the data page to be read has been marked as potentially not durable. If the data page is marked, then the transaction manager, at step 608, checks whether the page has already been made durable by another durable reader. If not, it immediately flushes the commit log to the data store at step 612, and takes steps 610 and 614 to indicate to other durable readers that this page is durable. Finally, it reads the data page at 616 and returns at step 618. On the other hand, if the transaction manager determines the read is not a durable read at step 602, or if it is a durable read but the data page is not marked at step 606, in both cases the transaction manager allows the application to immediately read the data page at step 616 and return at step 618.

[0040] Referring to Fig 7, at a later point in time, upon entry 700 the data manager is requested 702 that page to be written to the persistent data store. This request might be triggered by an application or a background process or by any other means. The data manager determines whether the page is marked 704. If it is not, it proceeds to writing the page to the data store 706. If the page is marked, data manager verifies whether the page has already been made durable by

another durable read application 708. If it is, then it clears the bit on the page 710 and writes the page to the data store 706 and then returns 712.

Conclusion

[0041] The various system, methods, and techniques described herein may be implemented with hardware or software or, where appropriate, with a combination of both. Thus, the methods and apparatus of the present invention, or certain aspects or portions thereof, may take the form of program code (i.e., instructions) embodied in tangible media, such as floppy diskettes, CD-ROMs, hard drives, or any other machine-readable storage medium, wherein, when the program code is loaded into and executed by a machine, such as a computer, the machine becomes an apparatus for practicing the invention. In the case of program code execution on programmable computers, the computer will generally include a processor, a storage medium readable by the processor (including volatile and non-volatile memory and/or storage elements), at least one input device, and at least one output device. One or more programs may be implemented in a high level procedural or object oriented programming language to communicate with a computer system. However, the program(s) can be implemented in assembly or machine language, if desired. In any case, the language may be a compiled or interpreted language, and combined with hardware implementations.

[0042] The methods and apparatus of the present invention may also be embodied in the form of program code that is transmitted over some transmission medium, such as over electrical wiring or cabling, through fiber optics, or via any other form of transmission, wherein, when the program code is received and loaded into and executed by a machine, such as an EPROM, a gate array, a programmable logic device (PLD), a client computer, a video recorder

or the like, the machine becomes an apparatus for practicing the invention. When implemented on a general-purpose processor, the program code combines with the processor to provide a unique apparatus that operates to perform the indexing functionality of the present invention.

[0043] While the present invention has been described in connection with the embodiments of the various figures, it is to be understood that other embodiments may be used or modifications and additions may be made to the described embodiment for performing the same function of the present invention without deviating there from. For example, while exemplary embodiments of the invention are described in the context of digital devices emulating the functionality of personal computers, the present invention is not limited to such digital devices, as described in the present application may apply to any number of existing or emerging computing devices or environments, such as a gaming console, handheld computer, portable computer, etc. whether wired or wireless, and may be applied to any number of such computing devices connected via a communications network, and interacting across the network. Furthermore, it should be emphasized that a variety of computer platforms, including handheld device operating systems and other application specific hardware/software interface systems, are herein contemplated, especially as the number of wireless networked devices continues to proliferate. Therefore, the present invention should not be limited to any single embodiment, but rather construed in breadth and scope in accordance with the appended claims.